INFORMATION PROCESSING SYSTEM ENABLING DYNAMICALLY LOADING OR REPLACING PROGRAM COMPONENT IN MEMORY ALLOCATED TO ACTIVATED PROCESS

5   BACKGROUND OF THE INVENTION

1) Field of the Invention

The present invention relates to an information processing system. In particular, the present invention relates to a distributed processing system realizing an

10 N-tier client-server environment. The present invention also relates to a method for updating a program component loaded in a server in an N-tier client-server environment.

2) Description of the Related Art

15 Recently, communication networks are required to have various functions. In order to provide various services to customers, manage operations, and maintain such communication networks, it is necessary to timely update versions of programs in the communication

20 networks. In particular, in backbone networks, it is impossible to stop the entire system or a process executed in the system. Therefore, the programs must be dynamically replaced during regular operations of the communication networks.

25 According to a conventional dynamic program replacement technique, a new version of a program is queued, and then replaces an old version of the program.

Thereafter, communication is restarted. According to another conventional dynamic program replacement technique, one or more requests for execution of an old version of a program are cancelled, and then the old

5   version of the program is replaced with a new version of the program. According to the above conventional techniques, it is not necessary to stop the entire process or the entire system. However, transaction processing is temporarily blocked or stopped.

10   On the other hand, distributed processing systems are widely used in the communication networks. The so-called client-server system is a typical example of the distributed processing system. Currently, the so-called N-tier client-server system is considered to be a next-

15   generation client-server system, and is under development. In the N-tier client-server system, more than one tier is formed above a client tier.

However, the conventional dynamic program replacement techniques are proposed for merely replacing

20   an old version of a program with a new version of the program. Therefore, the conventional dynamic program replacement techniques cannot achieve dynamic replacement of a plurality of old versions of a plurality of software components distributed over a

25   plurality of computers forming an N-tier client-server system, with new versions of the plurality of software components, so as to maintain synchronization between

the versions of the programs in the plurality of computers during the replacement.

SUMMARY OF THE INVENTION

5   An object of the present invention is to provide an information processing system which enables efficient, dynamic loading or replacement of a program component without service interruption.

Another object of the present invention is to
10   provide a distributed processing system realizing an N-tier client-server environment, and enabling efficient, dynamic loading or replacement of a distributed object without service interruption.

A further object of the present invention is to
15   provide a method for updating a program component loaded in a server in an N-tier client-server environment, whereby a program can be dynamically and efficiently loaded into the server, or a program loaded in the server can be dynamically and efficiently replaced,
20   without service interruption.

(1) According to the first aspect of the present invention, there is provided an information processing system including at least one terminal apparatus and at least one program execution apparatus. Each of the at
25   least one terminal apparatus comprises a message transmitting unit which transmits a message containing version information indicating a program version. Each

-3-

of the at least one program execution apparatus comprises a message receiving unit which receives the message containing version information indicating a program version, from one of the at least one terminal apparatus, a program storing unit which stores one or more program components, a pre-transfer information management table which holds information on the one or more program components stored in the program storing unit, a program memory unit which is allocated to an activated process, and temporarily stores at least one program component transferred from the program storing unit, a post-transfer information management table which holds information on the at least one program component stored in the program memory unit, and a program executing unit which dynamically links one of the one or more program components corresponding to the version information contained in the message received by the message receiving unit, to the program memory unit, so as to enable execution of the one of the one or more program components in the process.

As described above, in the information processing system according to the first aspect of the present invention, a message containing information on a program version is transmitted to a program execution apparatus, and a program component corresponding to the information on the program version is dynamically linked to an activated process in the program execution apparatus so

-4-

that the program can be executed. Therefore, a new version of software can be loaded into the program execution apparatus during execution of another version of software for transaction processing. That is, software can be dynamically and efficiently loaded into the program execution apparatus, or software loaded in the program execution apparatus can be dynamically and efficiently replaced, without stopping transaction.

The information processing system according to the first aspect of the present invention may have one or any possible combination of the following additional features (i) to (vii).

(i) The post-transfer information management table may indicate for each of the at least one program component a reference count which indicates the number of executions in which the each of the at least one program component are currently referred to.

(ii) When the information processing system according to the first aspect of the present invention has the above feature (i), the program executing unit may remove one of the at least one program component from the program memory unit when the reference count for the one of the at least one program component is zero, and information on a newer version of the one of the at least one program component is included in the post-transfer information management table.

(iii) The message received by the message

receiving unit may further contain an evaluation flag which indicates whether or not a program component is under evaluation.

(iv) When the information processing system according to the first aspect of the present invention has the above feature (iii), the program executing unit may execute the program component for which the evaluation flag is contained in the message, in a separate process, when the evaluation flag indicates that the program component for which the evaluation flag is contained in the message is under evaluation.

(v) The post-transfer information management table may indicate for each of the at least one program component an evaluation count which indicates the number of evaluations which are currently performed on the each of the at least one program component.

(vi) When the information processing system according to the first aspect of the present invention has the above feature (v), the program executing unit may terminate execution of one of the at least one program component in a separate process when the evaluation count corresponding to the one of the at least one program component is zero.

(vii) The information processing system according to the first aspect of the present invention may further comprise a management apparatus which sets and manages the one or more program components stored in

-6-

the program storing unit and the information on the one or more program components held in the pre-transfer information management table.

(2) According to the second aspect of the present invention, there is provided a terminal apparatus comprising: an interface unit which controls operations of inputting and outputting software; and a message transmitting unit which transmits a message containing version information indicating a program version.

The above message may further contain an evaluation flag which indicates whether or not a program component is under evaluation.

(3) According to the third aspect of the present invention, there is provided a program execution apparatus comprising: a message receiving unit which receives a message containing version information indicating a program version; a program storing unit which stores one or more program components; a pre-transfer information management table which holds information on the one or more program components stored in the program storing unit; a program memory unit which is allocated to an activated process, and temporarily stores at least one program component transferred from the program storing unit; a post-transfer information management table which holds information on the at least one program component stored in the program memory unit; and a program executing unit which dynamically links one

-7-

of the one or more program components corresponding to the version information contained in the message received by the message receiving unit, to the program memory unit, so as to enable execution of the one of the

5   one or more program components.

The terminal apparatus according to the second aspect of the present invention and the program execution apparatus according to the third aspect of the present invention are constituents of the information

10   processing system according to the first aspect of the present invention.

The program execution apparatus according to the third aspect of the present invention may have one or any possible combination of the aforementioned

15   additional features (i) to (vii).

(4) According to the fourth aspect of the present invention, there is provided a distributed processing system including at least one client apparatus and a plurality of server apparatuses, and realizing an N-tier

20   client-server environment. Each of the at least one client apparatus comprises a client stub processing unit which executes stub processing of a first message which contains version information indicating a program version. Each of the plurality of server apparatuses

25   comprises a server skeleton processing unit which executes skeleton processing of the first message containing version information indicating a program

-8-

version, a distributed-object storing repository which stores one or more distributed objects, a pre-transfer information management table which holds information on the one or more distributed objects stored in the distributed-object storing repository, a memory which is allocated to an activated process, and temporarily stores at least one distributed object transferred from the distributed-object storing repository, a post-transfer information management table which holds information on the at least one distributed object stored in the memory, a distributed object execution control unit which dynamically links one of the one or more distributed objects corresponding to the version information contained in the first message of which skeleton processing is executed by the server skeleton processing unit, to the memory, so as to enable execution of at least one function in the one of the one or more distributed objects, and a server stub processing unit which executes stub processing of a second message containing the version information so as to transmit the second message to another of the plurality of server apparatuses.

As described above, in the distributed processing system according to the fourth aspect of the present invention, a message containing information on a program version is transmitted to from a client apparatus to a server apparatus, and a program component corresponding

-9-

to the information on the program version is dynamically linked to an activated process in the server apparatus so that the program can be executed. Therefore, a new version of software can be loaded into the server apparatus during execution of another version of software for transaction processing. That is, software can be dynamically and efficiently loaded into the server apparatus, or software loaded in the server apparatus can be dynamically and efficiently replaced, without stopping transaction.

The distributed processing system according to the fourth aspect of the present invention may have one or any possible combination of the following additional features (viii) to (xiv).

(viii) The post-transfer information management table may indicate for each of the at least one distributed object a reference count which indicates the number of executions in which the each of the at least one distributed object are currently referred to.

(ix) When the distributed processing system according to the fourth aspect of the present invention has the above feature (viii), the distributed object execution control unit may remove one of the at least one distributed object from the memory when the reference count for the one of the at least one distributed object is zero, and information on a newer version of the one of the at least one distributed

object is included in the post-transfer information management table.

(x) Each of the first and second messages may further contain an evaluation flag which indicates whether or not a distributed object is under evaluation.

(xi) When the distributed processing system according to the fourth aspect of the present invention has the above feature (x), the distributed object execution control unit may execute the distributed object for which the evaluation flag is contained in the first message, in a separate process, when the evaluation flag indicates that the program component for which the evaluation flag is contained in the first message is under evaluation.

(xii) The post-transfer information management table may indicate for each of the at least one distributed object an evaluation count which indicates the number of evaluations which are currently performed on the each of the at least one distributed object.

(xiii) When the distributed processing system according to the fourth aspect of the present invention has the above feature (xii), the distributed object execution control unit may terminate execution of one of the at least one distributed object in a separate process when the evaluation count corresponding to the one of the at least one distributed object is zero.

-11-

(xiv) The distributed processing system according to the fourth aspect of the present invention may further comprise a management server which sets and manages the one or more distributed objects stored in

5 the distributed-object storing repository and the information on the one or more distributed objects held in the pre-transfer information management table.

(5) According to the fifth aspect of the present invention, there is provided a client apparatus

10 comprising: an interface unit which controls operations of inputting and outputting software; and a client stub processing unit which executes stub processing of a message which contains version information indicating a program version.

15 The above message may further contain an evaluation flag which indicates whether or not a distributed object is under evaluation.

(6) According to the sixth aspect of the present invention, there is provided a server apparatus

20 comprising: a server skeleton processing unit which executes skeleton processing of a first message containing version information indicating a program version; a distributed-object storing repository which stores one or more distributed objects; a pre-transfer

25 information management table which holds information on the one or more distributed objects stored in the distributed-object storing repository; a memory which is

allocated to an activated process, and temporarily stores at least one distributed object transferred from the distributed-object storing repository; a post-transfer information management table which holds information on the at least one distributed object stored in the memory; a distributed object execution control unit which dynamically links one of the one or more distributed objects corresponding to the version information contained in the first message of which skeleton processing is executed by the server skeleton processing unit, to the memory, so as to enable execution of at least one function in the one of the one or more distributed objects; and a server stub processing unit which executes stub processing of a second message containing the version information so as to transmit the second message to another server apparatus.

The client apparatus according to the fifth aspect of the present invention and the server apparatus according to the sixth aspect of the present invention are constituents of the distributed processing system according to the fourth aspect of the present invention.

The server apparatus according to the sixth aspect of the present invention may have one or any possible combination of the aforementioned additional features (viii) to (xiv).

(7) According to the seventh aspect of the present

invention, there is provided a method for updating a program component loaded in a server in an N-tier client-server environment, comprising the steps of: (a) storing in the server one or more program components;

5 (b) holding information on the one or more program components stored in the server, in a pre-transfer information management table; (c) transmitting a first message containing version information indicating a program version, from a client to the server; (d)

10 receiving the first message by the server; (e) dynamically linking one of the one or more program components corresponding to the version information contained in the first message, to a memory which is allocated to an activated process in the server, so as

15 to enable execution of the one of the one or more program components in the process; (f) holding in a post-transfer information management table information on at least one program component stored in the memory; and (g) transmitting to another server a second message

20 containing the version information.

As described above, in the method according to the seventh aspect of the present invention, a message containing information on a program version is transmitted from a client to a server, and a program

25 component corresponding to the information on the version of the program is dynamically linked to a process, and executed. Therefore, a new version of

- 14 -

software can be loaded during execution of another version of software for transaction processing. Therefore, a new version of software can be loaded into the server during execution of another version of software for transaction processing. That is, software can be dynamically and efficiently loaded into the server, or software loaded in the server can be dynamically and efficiently replaced, without stopping transaction.

The method according to the seventh aspect of the present invention may have one or any possible combination of the following additional features (xv) to (xxi).

(xv) The post-transfer information management table may indicate for each of the at least one program component a reference count which indicates the number of executions in which the each of the at least one program component are currently referred to.

(xvi) When the method according to the seventh aspect of the present invention has the above feature (xv), the method may further comprise the step of (h) removing one of the at least one program component from the memory when the reference count for the one of the at least one program component is zero, and information on a newer version of the one of the at least one program component is included in the post-transfer information management table.

(xvii) Each of the first and second messages may further contain an evaluation flag which indicates whether or not a program component is under evaluation.

(xviii) When the method according to the seventh aspect of the present invention has the above feature (xvii), in the step (f), the program component for which the evaluation flag is contained in the first message may be executed in a separate process when the evaluation flag indicates that the program component for which the evaluation flag is contained in the first message is under evaluation.

(xix) The post-transfer information management table may indicate for each of the at least one program component an evaluation count which indicates the number of evaluations which are currently performed on the each of the at least one program component.

(xx) When the method according to the seventh aspect of the present invention has the above feature (xix), the method may further comprise the step of (i) terminating execution of one of the at least one program component in a separate process when the evaluation count corresponding to the one of the at least one program component is zero.

(xxi) The method according to the seventh aspect of the present invention may further comprise the step of (j) setting and managing the one or more program

components stored in the server and the information on the one or more program components held in the pre-transfer information management table, by using a management apparatus.

5   (8) According to the eighth aspect of the present invention, there is provided a product for use with a program execution apparatus. When the product is used with the program execution apparatus, the product is able to output control information which directs the

10   program execution apparatus to comprise: a message receiving unit which receives a message containing version information indicating a program version; a program storing unit which stores one or more program components; a pre-transfer information management table

15   which holds information on the one or more program components stored in the program storing unit; a program memory unit which is allocated to an activated process, and temporarily stores at least one program component transferred from the program storing unit; a post-

20   transfer information management table which holds information on the at least one program component stored in the program memory unit; and a program executing unit which dynamically links one of the one or more program components corresponding to the version information

25   contained in the message received by the message receiving unit, to the program memory unit, so as to enable execution of the one of the one or more program

components.

The product according to the eighth aspect of the present invention may have one or any possible combination of the aforementioned additional features

5   (i) to (vii).

(9) According to the ninth aspect of the present invention, there is provided a product for use with a server apparatus. When the product is used with the server apparatus, the product is able to output control

10   information which directs the server apparatus to comprise: a server skeleton processing unit which executes skeleton processing of a first message containing version information indicating a program version; a distributed-object storing repository which

15   stores one or more distributed objects; a pre-transfer information management table which holds information on the one or more distributed objects stored in the distributed-object storing repository; a memory which is allocated to an activated process, and temporarily

20   stores at least one distributed object transferred from the distributed-object storing repository; a post-transfer information management table which holds information on the at least one distributed object stored in the memory; a distributed object execution

25   control unit which dynamically links one of the one or more distributed objects corresponding to the version information contained in the first message of which

skeleton processing is executed by the server skeleton processing unit, to the memory, so as to enable execution of at least one function in the one of the one or more distributed objects; and a server stub

5   processing unit which stub processing of a second message containing the version information so as to transmit the second message to another server apparatus.

The product according to the ninth aspect of the present invention may have one or any possible

10   combination of the aforementioned additional features (viii) to (xiv).

(10) According to the tenth aspect of the present invention, there is provided a product for use with a server in an N-tier client-server environment for

15   updating a program component loaded in the server. When the product is used with the server, the product is able to output control information which directs the server to execute a method comprising the steps of: (a) storing in the server one or more program components; (b)

20   holding information on the one or more program components stored in the server, in a pre-transfer information management table; (c) receiving a first message containing version information indicating a program version; (d) dynamically linking one of the one

25   or more program components corresponding to the version information contained in the first message, to a memory which is allocated to an activated process in the server,

so as to enable execution of the one of the one or more program components in the process; (e) holding in a post-transfer information management table information on at least one program component stored in the memory; and (f) transmitting to another server a second message containing the version information.

The product according to the tenth aspect of the present invention may have one or any possible combination of the aforementioned additional features (xv) to (xxi).

The above and other objects, features and advantages of the present invention will become apparent from the following description when taken in conjunction with the accompanying drawings which illustrate preferred embodiment of the present invention by way of example.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings:

Fig. 1 is a diagram illustrating a basic construction of an information processing system 1 according to the present invention;

Fig. 2 is a flow diagram illustrating the operations of the information processing system 1;

Fig. 3 is a diagram illustrating the construction of the information processing system as the first embodiment of the present invention;

Fig. 4 is a diagram illustrating the details of the construction of each of the server apparatuses 40-1 to 40-n;

Fig. 5 is a diagram illustrating an example of a format of the message;

Fig. 6 is a diagram illustrating an example of a definition of a message in the IDL (Interface Definition Language);

Figs. 7(A) and 7(B) are diagrams illustrating examples of mapping from the IDL definition to the programming language C++;

Fig. 8 is a diagram illustrating an example of the component-version-information definition file T1';

Fig. 9 is a diagram illustrating an example of a hierarchy of directories of the repositories;

Fig. 10 is a diagram illustrating an example of the component-version-information management table T2';

Fig. 11 is a flow diagram illustrating the operations of the distributed-object call control unit 43;

Fig. 12 is a diagram illustrating an example of a component-version-information management table T2-1;

Fig. 13 is a flow diagram illustrating the operation of releasing the memory from an old version of a distributed object (i.e., a behavior-processing common library), by the distributed-object call control unit 43 in the second embodiment;

Fig. 14 is a diagram illustrating an example of a format of a message 2-1 which is transmitted between a client and a plurality of server apparatuses;

Fig. 15 is a diagram illustrating an example of a definition of a message in the IDL (Interface Definition Language), which includes the evaluation flag 2d as well as the version information 2b;

Fig. 16 is a diagram illustrating an example of mapping from the IDL definition to the programming language C++;

Fig. 17 is a flow diagram illustrating the operation of handling a behavior-processing common library, which is under evaluation, by the distributed-object call control unit 43 in the third embodiment;

Fig. 18 is a diagram illustrating an example of a component-version-information management table T2-2;

Fig. 19 is a flow diagram illustrating the operations of controlling the evaluation count T2g and execution of at least one function in a version of a behavior-processing common library under evaluation, by the distributed-object call control unit 43 in the fourth embodiment; and

Fig. 20 is a flow diagram illustrating a sequence of operations performed in each server apparatus for changing versions of distributed objects in an N-tier client-server system.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the present invention are explained in detail below with reference to drawings.

5   (1) Functions of Program Replacement Systemes

Fig. 1 is a diagram illustrating a basic construction of an information processing system 1 according to the present invention. The information processing system 1 comprises a terminal 10 and a

10   program execution apparatus 20. The program execution apparatus 20 and the terminal 10 are connected through a network 100.

The message transmitting unit 11 in the terminal 10 transmits a transaction message (hereinafter called a

15   message) including version information, which indicates a version of an application program (hereinafter called a program) with which a currently loaded version of the application program is to be replaced. In the example of Fig. 1, the message which contains version information

20   indicating a version number "1.03" is transmitted from the terminal 10 to the program execution apparatus 20. A newer version is indicated by a greater version number. Although not shown, the terminal 10 further comprises an interface unit which controls input and output of

25   software.

The program storage unit 21 stores at least one program for loading or replacement. The pre-transfer

information management table T21 stores information on the at least one program stored in the program storage unit 21, where the information includes the version, the name, and the like of each of the at least one program. The program memory unit 22 is a memory used in a process Pa, and temporarily stores at least one program which is transferred from the program storage unit 21. In the example of Fig. 1, the program memory unit 22 stores versions "1.01" to "1.03" of a program. The post-transfer information management table T22 stores information on the at least one program stored in the program memory unit 22.

The program execution unit 23 dynamically links a version of a program corresponding to the version information contained in the message, to the program memory unit 22 which stores another version of the program currently running in the process Pa. In this specification, the dynamic linkage of a program to a memory means transfer of the program to the memory without stopping transaction processing or the like which is currently conducted by executing another program currently loaded in the memory, so that the linked program becomes executable.

For example, it is assumed that the versions "1.01" and "1.02" of the program have already been transferred from the program storage unit 21 to the program memory unit 22. In this case, when the program

execution apparatus 20 receives a message which contains version information indicating the version "1.03," the program execution unit 23 accesses the pre-transfer information management table T21 and the post-transfer

5 information management table T22 to recognize the location of the version "1.03" of the program, as explained later with reference to Fig. 2. Since, in this case, the version "1.03" of the program is not linked to the program memory unit 22, the program execution unit

10 23 transfers the version "1.03" of the program to the program memory unit 22. Thus, the version "1.03" of the program is linked to the process Pa, and thus the new version of the program can be executed.

As described above, according to the present

15 invention, it is possible to dynamically incorporate a new program in a currently running process. That is, a program component can be replaced without stopping transaction processing in the currently running process.

As another example, when a user requests execution

20 of the version "1.01" of the program during running of the version "1.03" of the program, a message which contains version information indicating the version number "1.01" is transmitted from the terminal 10 to the program execution apparatus 20. Thus, the program

25 execution unit 23 can call the version "1.01" of the program which is stored in the program memory unit 22, into execution.

In addition, the network management apparatus 50 is connected to the network 100. The network management apparatus 50 manages and maintains apparatuses connected to the network 100. For example, the network management

5   apparatus 50 sets and manages the programs stored in the program storage unit 21 and the information stored in the pre-transfer information management table T21.

The operations of the information processing system 1 are explained below with reference to Fig. 2,

10   which is a flow diagram illustrating the operations of the information processing system 1.

In step S1, the program execution apparatus 20 receives a message containing version information A which is transmitted from the terminal 10. In step S2,

15   the program execution unit 23 determines whether or not the post-transfer information management table T22 stores the version information A. When yes is determined in step S2, the operation goes to step S3. When no is determined in step S2, the operation goes to step S4. In

20   step S3, the program execution unit 23 executes the program corresponding to the version information A. In step S4, the program execution unit 23 refers to the pre-transfer information management table T21, and determines whether or not the program storage unit 21

25   stores the version information A. When yes is determined in step S4, the operation goes to step S6. When no is determined in step S5. In step S5, the program execution

apparatus 20 transmits an error message to the terminal
10. In step S6, the program execution unit 23
dynamically links the program corresponding to the
version information A stored in the program storage unit
5   21, to the program memory unit 22. In step S7, the
program execution unit 23 updates the contents of the
post-transfer information management table T22. In step
S8, the program execution unit 23 executes the program
corresponding to the version information A.
10

(2) First Embodiment

Fig. 3 is a diagram illustrating the construction
of the information processing system as the first
embodiment of the present invention. In the first
15   embodiment, the present invention is applied to a
distributed processing system realizing an N-tier
client-server environment. That is, the distributed
processing system 1a of Fig. 3 comprises a client
apparatus 30 and a plurality of server apparatuses 40-1
20   to 40-n which are arranged on a network 200 so as to
form an N-tier client-server architecture in which
basically distributed object computing is performed. In
addition, a distributed object communication
infrastructure called the Object Request Broker (ORB) is
25   provided on the network 200. The Object Request Broker
(ORB) in the construction of Fig. 3 is the Common Object
Request Broker Architecture (CORBA), DCOM (Distributed

Component Object Model (CORBA), or the like. In the following explanations, it is assumed that the ORB is CORBA.

In the construction of Fig. 3, first, a client application is downloaded from the Web server 60 to the client apparatus 30, where the Web server 60 is connected to the network 200. Thereafter, distributed objects are delivered through the ORB to the client apparatus 30 and the plurality of server apparatuses 40-1 to 40-n, and loading or replacement of distributed objects is performed through the ORB, where the distributed objects are each a program unit such as an application program, a library, or the like.

The client stub unit 31 in the client apparatus 30 executes stub processing of a message which contains version information indicating a version of at least one distributed object, where the stub processing is processing performed in a client apparatus for transmitting a message or a distributed object from the client apparatus to a server apparatus through the ORB. In addition, the client apparatus 30 comprises an interface unit (not shown) for controlling operations of inputting and outputting software.

In each of the plurality of server apparatuses 40-1 to 40-n, the server skeleton unit 44 executes skeleton processing of the message transmitted from the client apparatus 30, where the skeleton processing is

processing performed in a server apparatus for receiving a message or a distributed object transmitted from a client apparatus through the ORB. The distributed-object storing repository 41 stores at least one distributed object for loading or replacement, and the pre-transfer information management table T1 stores information on the at least one distributed object stored in the distributed-object storing repository 41. Details of the information stored in the pre-transfer information management table T1 are explained later with reference to Fig. 8. The memory 42 is a memory which is allocated to a process P1, and temporarily stores at least one distributed object transferred from the distributed-object storing repository 41. The post-transfer information management table T2 stores information on the at least one distributed object stored in the memory 42. Details of the information stored in the post-transfer information management table T2 are explained later with reference to Fig. 10. The distributed-object call control unit 43 dynamically links a distributed object corresponding to the version information contained in the received message, to the memory 42 allocated to the process P1 in which another version of the distributed object or another distributed object is currently executed. Then, the distributed-object call control unit 43 calls a function in the linked distributed object into execution. The server stub unit

45 executes stub processing of a message containing version information when a first one of the plurality of server apparatuses 40-1 to 40-n to which the server stub unit 45 belongs sends a request to a second one of the

5   plurality of server apparatuses 40-1 to 40-n. For example, the request is sent for changing a version of a distributed object in the second one of the plurality of server apparatuses 40-1 to 40-n, or calling a distributed object loaded in the second one of the

10 plurality of server apparatuses 40-1 to 40-n. That is, in this case, the first server apparatus acts a client.

In addition, the management server 50a is connected to the network 200. The management server 50a manages and maintains the apparatuses connected to the

15 network 200. For example, the network management apparatus 50a sets and manages the distributed objects stored in the distributed-object storing repository 41 and the information stored in the pre-transfer information management table T1.

20 For example, when versions "1.01" and "1.02" of a distributed object are stored in the memory 42, and the server apparatus 40-1 receives a message which contains version information indicating the version "1.03," the distributed-object call control unit 43 accesses the

25 pre-transfer information management table T1 and the post-transfer information management table T2 to recognize the location of the version "1.03" of the

distributed object, as explained later with reference to Fig. 11. Thus, the version "1.03" of the distributed object is linked to the process P1, and the new version of the distributed object can be executed. Next, the server stub unit 45 in the server apparatus 40-1 produces a message containing the version information "1.03," and sends the message to the server apparatus 40-2 so that the version "1.03" of the distributed object is also loaded into the server apparatus 40-2. Then, the server apparatus 40-2 also sends a message containing the version information "1.03" to the next server apparatus 40-3. By repeating a similar operation, the version "1.03" of the distributed object can be loaded into all of the server apparatuses 40-1 to 40-n connected to the network 200.

As described above, according to the present invention, it is possible to dynamically replace an old version of a distributed object in each of a plurality of server apparatuses with a new version of the distributed object, during transaction processing which is currently conducted by executing a program currently loaded in the memory. In addition, during loading or replacement of a version of distributed objects in server apparatuses, synchronization between the plurality of server apparatuses is maintained. After the loading or replacement, the transaction processing can be performed by execution of the program including the

loaded or replaced distributed object. That is, distributed objects in all of the server apparatuses connected to the network can be loaded or replaced automatically and efficiently without stopping transaction processing realized by the currently running program. Thus, reliability and quality of the system can be increased.

Details of the construction of each of the server apparatuses 40-1 to 40-n are explained below with reference to Fig. 4. In Fig. 4, the server apparatus 40 corresponds to each of the server apparatuses 40-1 to 40-n. In Fig. 4, the skeleton loading mechanism 44a and the skeleton processing unit 44b realize the server skeleton unit 44 in Fig. 3, and the stub loading mechanism 45a and the stub processing unit 45b realize the server stub unit 45 in Fig. 3. In addition, the component-version-information definition file T1' in Fig. 4 corresponds to the pre-transfer information management table T1 in Fig. 3, and the component-version-information management table T2' in Fig. 4 corresponds to the post-transfer information management table T2 in Fig. 3.

The skeleton storing repository 46 stores a common library for skeleton processing, which is hereinafter called a skeleton-processing common library. The skeleton loading mechanism 44a accesses the component-version-information definition file T1', and recognizes

a name of a process stored in the component-version-information definition file T1'. Then, the skeleton loading mechanism 44a calls a skeleton-processing common library La corresponding to the recognized name of the

5   process, which is stored in the skeleton storing repository 46. Thereafter, the skeleton processing unit 44b, which is included in the skeleton-processing common library La, performs processing for receiving a message. The distributed-object storing repository 41 stores as

10   at least one distributed object at least one common library for behavior processing (i.e., at least one common library each containing functions). Hereinafter, each common library for behavior processing is called a behavior-processing common library.

15   The distributed-object call control unit 43 accesses the component-version-information definition file T1' and the component-version-information management table T2' in order to recognize the location of the distributed object corresponding to version

20   information contained in the received message (as illustrated in Fig. 11). When the distributed object is not currently linked to the memory 42, the distributed-object call control unit 43 transfers a behavior-processing common library, from the distributed-object

25   storing repository 41 to the memory 42. Then, the distributed-object call control unit 43 calls at least one function defined in the distributed object (i.e., at

least one function included in the behavior-processing common library) into execution. In addition, the distributed-object call control unit 43 holds and manages the status of the dynamic linkage and a pointer
5   to each function in the distributed object.

The stub storing repository 47 stores at least one common library for server stub processing, which is hereinafter called a server-stub-processing common library. When a message is sent from the server
10   apparatus (first server apparatus) to which the stub storing repository 47 belongs, to another server apparatus (second server apparatus), e.g., for invoking a distributed object in the second server apparatus, the stub loading mechanism 45a in the first server apparatus
15   calls a server-stub-processing common library Lb, which is stored in the stub storing repository 47. Then, the stub processing unit 45b, which is included in the server-stub-processing common library Lb, performs processing for transmitting a message to the second
20   server apparatus.

Specifically, when a request for processing by a server application in the second server apparatus arises during behavior processing (i.e., execution of a function) of a distributed object in the first server
25   apparatus, the stub loading mechanism 45a is triggered in the first server apparatus. Then, in order to enable communication with the server application in the second

server apparatus, an appropriate server-stub-processing common library Lb is acquired from the stub storing repository 47 in the first server apparatus, and dynamically linked to a memory. Then, the stub

5 processing unit 45b in the first server apparatus executes processing for establishing a connection with the server application in the second server apparatus. At this time, the behavior processing of the distributed object is suspended until the above connection is

10 established. When the connection is established, the distributed object packs the version information in a message, and sends the message to the second server apparatus. By repeating the above operations, it is possible to change the versions of distributed objects

15 in a plurality of server apparatuses 40-1 to 40-n, or deliver distributed objects to the plurality of server apparatuses 40-1 to 40-n, so as to maintain synchronization between the distributed objects or the versions of the distributed objects in the plurality of

20 server apparatuses.

Generally, a plurality of processes P1 to Pn can be generated. Each of the plurality of processes P1 to Pn can communicate with each of the skeleton storing repository 46, the stub storing repository 47, the

25 distributed-object storing repository 41, and the component-version-information definition file T1' through the operating system 48.

The common libraries are delivered to the skeleton storing repository 46, the stub storing repository 47, and the distributed-object storing repository 41 by the management server 50a, and the skeleton storing repository 46, the stub storing repository 47, and the distributed-object storing repository 41 are centrally controlled by the management server 50a.

Fig. 5 is a diagram illustrating an example of a format of the message. The format 2 in Fig. 5 is comprised of a function name 2a of a distributed object, version information 2b, and parameters 2c. In this example, the version information 2b is a version number of the entire system, and is comprised of a major number and a minor number.

Fig. 6 is a diagram illustrating an example of a definition of a message in the IDL (Interface Definition Language), which is a language used for defining interfaces of distributed objects. As illustrated in Fig. 6, in all interfaces, version information is defined as "in" parameters, and the other parameters follow the version information.

Figs. 7(A) and 7(B) are diagrams illustrating examples of mapping from the IDL definition to the programming language C++, where Fig. 7(A) corresponds to the version "1.01," and Fig. 7(B) corresponds to the version "1.02". As illustrated in Figs. 7(A) and 7(B), according to the present invention, the class XXX

corresponding to the version "1.01" and the class XXX corresponding to the version "1.02" are defined in different namespaces "XXX 1_01" and "XXX 1_02," respectively, when written in the programming language

5   C++. Thus, the names of functions and variables in different versions of distributed objects do not clash, and therefore more than one version of the behavior-processing common library can be dynamically linked to a memory allocated to a process.

10   Fig. 8 is a diagram illustrating an example of the component-version-information definition file T1'. The component-version-information definition file T1' includes version information T1a indicating a version of the entire system, a name T1b of a behavior-processing

15   common library, version information T1c indicating the version of the behavior-processing common library, and a process name T1d.

        The version information T1a corresponds to the version information contained in the request message

20   sent from the client. For example, in Fig. 8, the names "libAPL-A.so" to "libAPL-X.so" of the common libraries correspond to the version information "1.01". In addition, the skeleton storing repository 46 and the stub storing repository 47 store common libraries which

25   have the same name and different contents. For example, a common library having the name "libAPL-A.so" stored in the skeleton storing repository 46 is a skeleton-

- 37 -

processing common library La for the process "process-1," and another common library having the same name "libAPL-A.so" stored in the stub storing repository 47 is a stub-processing common library Lb for the process

5 "process-1".

The version information T1c indicates the version of each distributed object (software component). Generally, the entire system is constituted by a combination of a plurality of versions of a plurality of

10 software components, and the relationship between the version (the version information T1c) of each software component (constituent) and the version of the entire system are indicated in the component-version-information definition file T1'. That is, the version of

15 each software component can be definitely determined based on the version of the entire system.

The process name T1d indicates a process to which each software component is to be linked. That is, the process name T1d indicates a process in which each

20 software component is to be executed.

Fig. 9 is a diagram illustrating an example of a hierarchy of directories of the repositories. As illustrated in Fig. 9, directories /etc, /bin, /skel, /stub, and /obj are arranged under the directories /opt

25 and /apt. The directory /etc includes the component-version-information definition file T1', the directory /bin includes the skeleton loading mechanism 44a, the

directory /skel includes the skeleton storing repository 46, and the directory /stub includes the stub storing repository 47. In addition, the directories /1.01, /1.02, and the like are arranged under the directory /obj, where the directories /1.01, /1.02, and the like include the respective versions of the distributed-object storing repositories 41.

Fig. 10 is a diagram illustrating an example of the component-version-information management table T2', which stores information on the at least one distributed object stored in the memory 42. In the example of Fig. 10, the component-version-information management table T2' indicates version information T2a indicating a version of the entire system, a name T2b of a behavior-processing common library, version information T2c indicating a version of the behavior-processing common library, a flag T2d, and an address T2e of a function table. The version information T2a, the name T2b of the behavior-processing common library, and the version information T2c are respectively similar to the version information T1a, the name T1b of the behavior-processing common library, and the version information T1c in Fig. 8. The flag T2d indicates Y when the corresponding version of the behavior-processing common library is loaded in the memory 42, and N when the corresponding version of the behavior-processing common library is not loaded in the memory 42. The address T2e indicates an

address of a function table which contains addresses of functions in the corresponding version of the behavior-processing common library. For example, the address of the function table t(1.01) which contains the addresses

5   of the functions corresponding to the version "1.01" of the behavior-processing common library is "0x00012345," and the address of the function table t(1.02) which contains the address of the function corresponding to the version "1.02" of the behavior-processing common

10   library is "0x00023456."

The operations of the distributed-object call control unit 43 are explained below with reference to Fig. 11, which is a flow diagram illustrating the operations of the distributed-object call control unit

15   43.

In step S10, when the server apparatus receives a message containing version information A, the distributed-object call control unit 43 determines whether or not the component-version-information

20   management table T2' contains the version information A. When yes is determined in step S10, the operation goes to step S11. When no is determined in step S10, the operation goes to step S12. In step S11, the distributed-object call control unit 43 calls at least

25   one function in a behavior-processing common library corresponding to the version information A, into execution. In step S12, the distributed-object call

control unit 43 refers to the component-version-information definition file T1' to determine whether or not the component-version-information definition file T1' contains the version information A. When yes is determined in step S12, the operation goes to step S14. When no is determined in step S12, the operation goes to step S13. In step S13, the server apparatus transmits an error message to the client. In step S14, the distributed-object call control unit 43 dynamically links the behavior-processing common library corresponding to the version information A and being stored in the distributed-object storing repository 41, to the memory 42. In step S15, the distributed-object call control unit 43 updates the contents of the component-version-information management table T2'. In step S16, the distributed-object call control unit 43 calls at least one function in a behavior-processing common library corresponding to the version information A, into execution.

As explained above, the distributed-object call control unit 43 is provided in each server apparatus (e.g., a server application loaded in each server apparatus), and dynamically links a behavior-processing common library corresponding to the version of the entire system which is contained in a received message, to a memory allocated to the corresponding process.

In addition, functions and variables in different

versions of a behavior-processing common library are defined in different namespaces, when the program is written in the programming language C++. Therefore, the names of functions and variables in different versions

5   of distributed objects do not clash. Thus, different versions of software can be concurrently linked to a memory allocated to the corresponding process, without stopping transaction.

Further, according to the present invention,

10 messages containing information on the version of the entire system are transmitted between a client application and a plurality of server applications. Therefore, it is possible to change the versions of distributed objects in the plurality of server

15 applications so as to maintain synchronization between the versions of the distributed objects in the plurality of server apparatuses even when the client and the plurality of server apparatuses form an N-tier client-server system.

20

(3) Second Embodiment

In the second embodiment of the present invention, an old version of a behavior-processing common library is automatically removed from a memory allocated to a

25 process in which at least one function in the old version of the behavior-processing common library is executed, i.e., the memory is automatically released

from the old version of the behavior-processing common library, when a newer version of the behavior-processing common library is loaded in the memory, and the old version of the behavior-processing common library is not

5   used.

Fig. 12 is a diagram illustrating an example of a component-version-information management table T2-1. The component-version-information management table T2-1 in the second embodiment is different from the component-version-information management table T2' in the first

10   embodiment in that the component-version-information management table T2-1 further contains a reference count T2f for each version of each behavior-processing common library. The reference count T2f for each version of

15   each behavior-processing common library indicates the number of current references (calls) to the behavior-processing common library corresponding to the version, i.e., the number of executions in which the version of the behavior-processing common library is currently

20   referred to (called). When the distributed-object call control unit 43 calls a function in a version of a behavior-processing common library, the distributed-object call control unit 43 increments the reference count T2f for the version of the behavior-processing common library by one. Thereafter, when the execution of

25   the called function is completed, i.e., when the operation returns from the execution of the called

function, the distributed-object call control unit 43 decrements the reference count T2f for the above version of the behavior-processing common library by one.

Fig. 13 is a flow diagram illustrating the operation of releasing the memory from an old version of a distributed object (i.e., a behavior-processing common library), by the distributed-object call control unit 43 in the second embodiment.

In step S20, when the distributed-object call control unit 43 completes execution of a function of a first version of the behavior-processing common library, the distributed-object call control unit 43 decrements the reference count T2f for the first version of the behavior-processing common library by one. In step S21, the distributed-object call control unit 43 determines whether or not the value of the reference count T2f for the first version of the behavior-processing common library is zero. When yes is determined in step S21, the operation goes to step S22. When no is determined in step S21, the operation goes to step S24. In step S22, the program execution unit 23 determines whether or not the component-version-information management table T2-1 contains version information indicating a newer version than the above first version. When yes is determined in step S22, the operation goes to step S23. When no is determined in step S22, the operation goes to step S24. In step S23, the distributed-object call control unit 43

removes the older (first) version of the behavior-processing common library from the memory 42, i.e., releases the memory 42 from the older (first) version of the behavior-processing common library. In step S24, the

5 distributed-object call control unit 43 continues processing.

As explained above, in the second embodiment, the memory is automatically released from an old version of a behavior-processing common library, when a newer

10 version of the behavior-processing common library is loaded in the memory, and the old version of the behavior-processing common library is not used. Therefore, the version of the behavior-processing common library loaded in the memory can be changed efficiently.

15

(4) Third Embodiment

In the third embodiment of the present invention, a separate process is created for executing a distributed object under evaluation. Fig. 14 is a

20 diagram illustrating an example of a format of a message 2-1 which is transmitted between a client apparatus and a plurality of server apparatuses. The format 2-1 of Fig. 14 is comprised of a function name 2a of a distributed object, version information 2b, parameters 2c, and an

25 evaluation flag 2d. The version information 2b indicates a version number of the entire system, and is comprised of a major number and a minor number. The evaluation

flag 2d indicates whether or not the corresponding version of the behavior-processing common library is under evaluation. Fig. 15 is a diagram illustrating an example of a definition of a message in the IDL (Interface Definition Language), which includes the evaluation flag 2d as well as the version information 2b. Fig. 16 is a diagram illustrating an example of mapping from the IDL definition to the programming language C++. As illustrated in Fig. 16, in the third embodiment, the class definition of Fig. 16 also includes the evaluation flag 2d as well as the version information 2b.

Fig. 17 is a flow diagram illustrating the operation of handling a behavior-processing common library, which is under evaluation, by the distributed-object call control unit 43 in the third embodiment.

In step S30, the distributed-object call control unit 43 determines whether or not the evaluation flag 2d contained in a received message, which also contains version information A indicating the version of the entire system, indicates that the corresponding behavior-processing common library is under evaluation. When yes is determined in step S30, the operation goes to step S31. When no is determined in step S30, the operation goes to step S10 in Fig. 11. In step S31, the distributed-object call control unit 43 determines whether or not the component-version-information definition file T1' contains the version information A.

When yes is determined in step S31, the operation goes to step S33. When no is determined in step S31, the operation goes to step S32. In step S32, the server apparatus 40 transmits an error message to the client.

5   In step S33, the distributed-object call control unit 43 creates a separate process, and employs a skeleton loading mechanism in the separate process. Then, the distributed-object call control unit 43 loads a skeleton-processing common library Lb corresponding to

10   the separate process and the behavior-processing common library corresponding to the version information A, into a memory allocated to the separate process. In step S34, the distributed-object call control unit 43 updates the contents of the component-version-information management

15   table T2'. In step S 35, the distributed-object call control unit 43 executes at least one function in the behavior-processing common library under evaluation in the separate process.

As explained above, in the third embodiment, a

20   distributed object under evaluation is executed in a separate process. Therefore, execution of a function in a version of a behavior-processing common library under evaluation does not affect execution of functions in the other behavior-processing common libraries, and thus the

25   quality of the replacement control of distributed objects can be improved.

-47-

(5) Fourth Embodiment

In the fourth embodiment of the present invention, execution, in a separate process, of a function included in a behavior-processing common library under evaluation is automatically terminated when the version of the behavior-processing common library under evaluation is loaded in the memory, and the server apparatus receives a notification of completion of the evaluation.

Fig. 18 is a diagram illustrating an example of a component-version-information management table T2-2. The component-version-information management table T2-2 in the fourth embodiment is different from the component-version-information management table T2-1 in the second embodiment in that the component-version-information management table T2-2 further contains an evaluation count T2g for each version of each behavior-processing common library. The evaluation count T2g for each version of each behavior-processing common library indicates the number of evaluations currently performed on the version of the behavior-processing common library. When the distributed-object call control unit 43 creates a separate process, and links a behavior-processing common library which is under evaluation, to a memory allocated to the separate process, the distributed-object call control unit 43 increments the evaluation count T2g by one. Thereafter, when the server apparatus receives a notification of completion of the evaluation

of the behavior-processing common library, the distributed-object call control unit 43 decrements the evaluation count T2g by one.

Fig. 19 is a flow diagram illustrating the operations of controlling the evaluation count T2g and execution of at least one function in a version of a behavior-processing common library under evaluation, by the distributed-object call control unit 43 in the fourth embodiment.

In step S40, when the distributed-object call control unit 43 receives a notification of completion of the evaluation of the above version of the behavior-processing common library, during execution of a function in the version of the behavior-processing common library, the distributed-object call control unit 43 decrements the evaluation count T2g by one. In step S41, the distributed-object call control unit 43 determines whether or not the evaluation count T2g is zero. When yes is determined in step S41, the operation goes to step S42. When no is determined in step S41, the operation goes to step S44. In step S42, the distributed-object call control unit 43 determines whether the flag T2d is Y or N, i.e., whether or not the above version of the behavior-processing common library under evaluation is dynamically linked to a memory which is allocated to a separate process created for the version of the behavior-processing common library. When

yes is determined in step S42, the operation goes to step S43. When no is determined in step S42, the operation goes to step S44. In step S43, the distributed-object call control unit 43 terminates the separate process. In step S44, the distributed-object call control unit 43 continues execution.

As explained above, in the fourth embodiment, when the process for evaluation becomes unnecessary, the process for evaluation is automatically terminated.

(6) Sequence of Operations

Fig. 20 is a flow diagram illustrating a sequence of operations performed in each server apparatus for changing versions of distributed objects in an N-tier client-server system.

In step S50, the server apparatus stores one or more program components for use in loading or replacement. In step S51, the server apparatus holds in a pre-transfer information management table information on the one or more program components. In step S52, a client transmits a message containing information on a program version. In step S53, the server apparatus receives the message. In step S54, the server apparatus dynamically links one of the one or more program components corresponding to the program version, to a memory allocated to a process in which a program component which is to be replaced is executed, and calls

the one of the one or more program components into execution. In step S55, the server apparatus holds in a post-transfer information management table information on at least one program component stored in the memory. In step S56, program components in the other server apparatuses in an N-tier client-server system are replaced in preparation for execution of program components in the other server apparatuses so as to maintain synchronization between program components loaded in the server apparatuses.

In particular, in the fourth embodiment of the present invention, the following operations are performed during execution of programs in each server apparatus.

(i) The post-transfer information management table further contains a reference count for each version of each program component. The reference count for each version of each program component is the number of current references to a function in the version of the program component (i.e., the number of executions in which a function in the version of the program component is currently referred to). A memory allocated to a process in which a first version of a program component is executed is automatically released from the first version of the program component, when a newer (second) version of the program component is loaded in the memory, and the first (older) version of the program component

is not used.

(ii) Each message transmitted from a client or a server apparatus to another server apparatus contains an evaluation flag as well as the version information, where the evaluation flag indicates whether or not a version of a program component is under evaluation. When each server apparatus receives a message which contains an evaluation flag indicating that a version of a program component is under evaluation, the server apparatus executes the version of the program component in a separate process.

(iii) The component-version-information management table further contains an evaluation count for each version of each program component, where the evaluation count for each version of each program component indicates the number of evaluations which are currently performed on the version of the program component. When the evaluation count for a version of a program component is zero in a server apparatus, the server apparatus terminates execution of the version of the program component in the separate process.

(iv) The program components for use in loading or replacement and the information on the program components are set and managed by the management server.

(7) Advantages

(i) As described above, according to the present

invention, new software components can be dynamically loaded, without stopping transaction, in an N-tier client-server system in which distributed object computing is performed.

5   (ii) It is possible to change step by step a plurality of versions of a plurality of software components which are distributed over a plurality of computers when more than one version of a software component may coexist in each computer.

10   (iii) An unnecessarily used memory portion is released from an old version of a software component when completion of transaction processing for a client using the old version of the software component is confirmed. Therefore, system resources can be

15   efficiently used.

(iv) When a software component is under evaluation, the software component is not executed in a process in which transaction processing is executed, and is instead executed in a separate process. Therefore, it is

20   possible to load a new software component in a server apparatus without decrease in reliability.

(v) When evaluation of a software component is completed, the software component is automatically loaded into a memory which is allocated to a process

25   created for transaction, and then transaction is started. In addition, when transaction processing executed in a separate process is completed, the unnecessary

(separate) process is terminated. Therefore, system resources can be efficiently used.

(vi) Software components are centrally controlled in a repository server, and software transfer is made. Therefore, it is possible to simplify and automate the operation of updating software, which is usually complicated in an N-tier client-server environment. In addition, the quality of the operation can be improved.

(8) Other Matters

(i) Although CORBA is used as an object request broker (ORB) in the above embodiments, any other object request broker can be used. In addition, although applications of the present invention to N-tier client-server systems are mainly explained in the above embodiments, the present invention can be applied to dynamic loading or replacement of software in any other information communication networks.

(ii) The operations and the functions of the server apparatus in each of the first to fourth embodiments can be realized by using at least one product with an information processing apparatus such as a computer, e.g., by installing a computer-readable medium in a computer. The product is such that when the product is used with an information processing apparatus, the product is able to output control information which directs the information processing apparatus to realize

one or more functions of the server apparatus in each of the first to fourth embodiments. The product may be a computer-readable medium which stores a program which directs the computer to realize one or more functions of the server apparatus in each of the first to fourth embodiments. The computer-readable medium may be a semiconductor storage device such as a ROM, a magnetic storage medium such as a floppy disc or a hard disk, or a CD-ROM, a CD-R, a DVD-ROM, a DVD-RAM, a DVD-R, or the like. Alternatively, the product may be a programmed hardware logic circuit such as a custom LSI, or realized by a combination of hardware and software. The above product can be put into the market. Otherwise, the program data which directs the computer to realize one or more functions of the server apparatus in each of the first to fourth embodiments can be transferred through a communication network from a storage device included in a computer system to another computer system. When executing the program in a computer system, for example, the program stored in a hard disk drive may be loaded into a main memory of the computer system.

(iii) The foregoing is considered as illustrative only of the principle of the present invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and applications shown and described, and accordingly, all

suitable modifications and equivalents may be regarded as falling within the scope of the invention in the appended claims and their equivalents.

(iv) In addition, all of the contents of the Japanese patent application, No.2000-077814 are incorporated into this specification by reference.